תכנות מונחה עצמים – תרגיל 1

Tic-Tac-Toe Tournament

תאריך ההגשה : 28.05.24 בשעה 23:55

שימו לב! תרגיל זה יש להגיש רק ביחידים (לא בזוגות).

0.הקדמה

בתרגיל זה נממש טורניר של משחקי איקס עיגול.

בגרסה שלנו הלוח יהיה בגודל $n \times n$, וסיבוב יחיד של משחק נגמר כאשר יש שחקן מנצח או כאשר לא נותרו משבצות ריקות בלוח. שחקן מוגדר כמנצח אם הוא השיג על הלוח רצף של k משבצות מסומנות בסימן שלו (X או O). הרצף יכול להיות מאונך, מאוזן או אלכסוני. כאשר n ו k הם משתנים עם ערכים שיבחר המשתמש, ועם הערכים הדיפולטיביים 4 ו-3 בהתאמה. נשים לב, שבשביל שיהיה משמעות למשחק, נדרוש $2 \leq k \leq n$.

בתרגיל יהיו שני סוגי שחקנים, שחקן אנושי שישחק את תורו דרך ה- `System.in`; ושחקן אוטומטי, שיפעל באופן אוטומטי לפי אסטרטגיה מוגדרת.

1.מהלך המשחק

1.1. בתחילת המשחק יבחר המשתמש שני סוגי שחקנים, השחקנים האלה יכולים להיות מאותו סוג - שניהם אוטומטיים/אנושיים, או מסוגים שונים - אחד אוטומטי ואחד אנושי.

1.2. השחקנים שנבחרו יערכו טורניר המורכב ממספר סיבובים שיגדיר המשתמש.
בכל סיבוב יחליפו השחקנים תפקיד כך שישחקו לסירוגין בסימנים איקס ועיגול, ולוח התוצאות יציג כמה סיבובים ניצח שחקן 1, כמה סיבובים ניצח שחקן 2 וכמה סיבובים הסתיימו בתיקו.

1.3. כמו כן, יבחר המשתמש משתנה שלישי שמגדיר האם הלוח ירונדר על המסך או לא. האפשרות לא להציג את הלוח שימושית במיוחד כשרוצים להריץ טורניר של מספר סיבובים בין שחקנים אוטומטיים.

2.שורת ההרצה

2.1. משתמש שרוצה להריץ את המשחק, יקליד את הפקודה הבאה:

```
java Tournament [round count] [size] [win_streak]
[render target: console/none]
[first player: human/whatever/clever/genius]
[[second player: human/whatever/clever/genius
```

לדוגמא, הפקודה הבאה תריץ טורניר של 10,000 סיבובים על לוח בגודל 4×4, ורצף ניצחון של 3, בין השחקנים `Mr. Whatever` ו `Mr. Clever` (ר׳ פירוט למטה לגבי סוגי השחקנים), ולא תדפיס את הלוח על המסך:

```
java Tournament 10000 4 3 none whatever clever
```

כדי לשחק שלושה משחקים בין שחקן אוטומטי למשתמש, הפקודה היא:

```
java Tournament 3 4 3 console whatever human
```

2.2. במקרה שבו תבחרו בערך `none` עבור רינדור הלוח ותגדירו גם שחקן אנושי, תקבלו בדיוק את מה שביקשתם: הלוח לא יודפס על המסך אבל כשיגיע תורו של המשתמש האנושי הוא עדיין יתבקש לבחור משבצת על לוח המשחק.

3.תקינות הקלט

שורת הרצה:

3.1. ניתן להניח שאת התוכנית מריצים בדיוק עם 6 ארגומנטים בסדר הנכון.

3.2. ניתן להניח שיתקבל מספר שלם וגדול מאפס עבור כמות הסבבים שצריך לשחק.

3.3. ניתן להניח שגודל הלוח הוא ספרה בין 2-9.

3.4. ניתן להניח שרצף הניצחון הוא ספרה בין 2-9 וקטן מגודל הלוח.

3.5. שגיאות בשורת ההרצה:

במידה ויש שגיאת הקלדה ("$Typo$") באחד משמות השחקנים <u>או</u> בסוג הרינדור, תודפס על המסך הודעה הבאה, וריצת התוכנית תסתיים ללא הרצת טורניר:

Usage: Please run the game again: java Tournament [round count] [size] [win_streak] [render target: console/none] [first player: human/whatever/clever/genius] [second player: human/whatever/clever/genius]

- דוגמא לפקודה עם שגיאת הקלדה בשם המשתמש:

```
java Tournament 3 4 3 console whtever human
```

- דוגמא לפקודה עם שגיאת הקלדה בסוג הרינדור:

```
java Tournament 3 4 3 oops human human
```

קלט של שחקן אנושי:

3.6. יש לוודא שהקלט ששחקן מכניס הוא תקין, כלומר בגבולות הלוח.
למשל - אם גודל הלוח הנבחר הוא 6, **לא תקין** ששחקן יבחר במשבצת בשורה **7** ועמודה **5** על גבי הלוח.
במקרה כזה, נא להדפיס שהקלט לא חוקי עם ההודעה הבאה (שימו לב שבסוף ההדפסה תהיה ירידת שורה):

Invalid mark position, please choose a different position.
Invalid coordinates, type again:

3.7. אם שחקן בוחר משבצת שיש עליו כבר סימון של שחקן כלשהו, כלומר המשבצת אינה blank, אז יש להדפיס (שימו לב שבסוף ההדפסה תהיה ירידת שורה) :

Mark position is already occupied.
Invalid coordinates, type again:

API.4

<u>בגדול:נציג את ה API של המשחק המורכב משני ממשקים, שלוש מחלקות משחק, שתי מחלקות רינדור, ארבע מחלקות שחקן, שתי מחלקות מפעל ומ $Enum$ אחד:</u>

- ממשקים:
  - `Renderer`
  - `Player`
- מחלקות:
  - `Board`
  - `VoidRenderer`
  - `ConsoleRenderer` – המימוש שלה נמצא ב-Moodle.
  - `KeyboardInput` – המימוש שלה נמצא ב-Moodle.
  - `HumanPlayer`
  - `CleverPlayer`
  - `WhateverPlayer`
  - `GeniusPlayer`
  - `Game`

- Tournament
- PlayerFactory
- RendererFactory
- ★ enum Mark

הערות:

- ❖ שימו לב שהמחלקה ConsoleRenderer מסופקת לכם במודל, ואין צורך לממש אותה בעצמכם.
- ❖ כל המתודות שנציג כחלק מה *API* יהיו בעלות מגדיר נראות פומבי (public).
- ❖ הגדירו פונקציות עזר פרטיות ושדות פרטיים לפי הצורך.

טיפ: התחילו קודם כל ביצירת הממשקים.

להלן תיאור מפורט של המחלקות הממשקים וה-enum בתרגיל:

4.1. Mark:

Enum בשם Mark ייצג את הסימונים על הלוח, והוא יוגדר כך:

```
enum Mark{BLANK, X, O}
```

והוא ישב בקובץ Mark.java .

בנוסף, עליכם ליצור מתודה בשם **toString** ב-Enum. מטרתה של המתודה היא המרת הסימון הנבחר ב-Enum ל-String. תוכלו להשתמש בה על מנת להדפיס את הסימון בקלות.

במקרה שה-Enum הנבחר הוא BLANK, ניתן להחזיר null.

חתימת המתודה היא:

```
String toString()
```

4.2. המחלקה Board :

אחראית על מצב הלוח: גודל הלוח, סימון משבצות ושמירת כל מה שסומן על הלוח. שימו לב כי מספר המשבצות על הלוח מתחיל ב-0.

רשימת המתודות של המחלקה:

מתודה	משמעות
`Board()`	בנאי דיפולטיבי, מגדיר לוח ריק חדש בגודל הדיפולטיבי.
`Board(int size)`	בנאי נוסף, מגדיר לוח ריק חדש בגודל $size * size$.
`int getSize()`	מחזירה את גודל הלוח, כלומר אורך שורה או אורך עמודה (הלוח תמיד בצורה של $n * n$.)
`boolean putMark(Mark mark, int row, int col)`	תנסה לסמן את המשבצת (row, col) ב $mark$ הנבחר. מחזירה $True$ אם"ם סימנה את המשבצת בהצלחה (לפי כללי המשחק הסטנדרטיים).
`Mark getMark(int row, int col)`	מחזירה את הסימון שיש במשבצת (row, col). במקרה של קבלת קואורדינטות לא חוקיות תחזיר $Mark.BLANK$

4.3. הממשק `Renderer` ומחלקות הרינדור :

ממשק שמייצג צורה להצגת מהלך של משחקי איקס עיגול על המסך.

מתודה	משמעות
`void renderBoard(Board board)`	קבל לוח והצג אותו לשיטתך.

4.3.1. המחלקה `ConsoleRenderer`:

את המחלקה הזו אתם אינכם ממשים או משנים. המימוש של מחלקה זו נמצא בMoodle הקורס, <u>אין לשנות או לערוך אותה.</u>

המחלקה מייצגת צורה של הצגה ויזואלית של מהלך משחק על הלוח על ידי הדפסות לטרמינל. ה API של המחלקה מכיל שתי מתודות:

- בנאי דיפולטיבי.
- מתודה עם החתימה: $void\ renderBoard(Board\ board)$, שמקבלת לוח ומציגה אותו על המסך.

4.3.2. המחלקה VoidRenderer :

המחלקה מממשת (implements) את הממשק Renderer, אבל מהווה מימוש ריק של Renderer שלמעשה לא מציג כלום על המסך, אך זה עדיין נחשב צורה של רינדור. בעזרת המחלקה הזו, ניתן להריץ משחק בין שחקנים שתממשו מבלי לראות הדפסות של הלוח על המסך.

4.4. הממשק Player ומחלקות השחקים:

ממשק שמייצג לוגיקה או אסטרטגיה מסוימות לביצוע תור במשחק בהינתן לוח מסוים.

משמעות	מתודה
מבצעת את האסטרטגיה של השחקן.	**void playTurn(Board board, Mark mark)**

ארבעת המחלקות GeniusPlayer, WhateverPlayer, CleverPlayer ו HumanPlayer יממשו את Player.

בנוסף לשחקן האנושי, תממשו שלושה שחקנים אוטומטיים. השחקנים האוטומטיים אמורים לתת תוצאה מסויימת, ואנו מצפים מכם לחשוב איך להשיג אותה.

- המהלכים של WhateverPlayer אקראיים.
- CleverPlayer תנצח את WhateverPlayer ברוב המוחלט של המשחקים.
- GeniusPlayer תנצח את CleverPlayer ברוב המוחלט של המשחקים.

במילים אחרות, אנו דורשים שתממשו לוגיקה לשחקנים האוטומטיים כך שברוב המוחלט של המשחקים יתקיים GeniusPlayer > CleverPlayer > WhateverPlayer. <u>הסבר מדויק על כמות הנצחונות רשום בתיאור המחלקות עצמן בסעיפים הבאים.</u>

שימו לב שהמחלקות המממשות את הממשק player לא תלויות במשחק מסוים; מופע אחד של שחקן יכול לקחת חלק במשחק אחד או בכמה, ועם סימונים שונים. כתוצאה מכך, הבנאי של כל אחד מהמחלקות שממשות את הממשק player לא דורשות שום קלט, היות ולא נדרש אף מידע לגבי המשחק או הלוח כדי לאתחל את השחקן.

4.4.1. המחלקה HumanPlayer :

המחלקה מייצגת שחקן אנושי. האחריות היחידה של מחלקה זו היא בקשת קלט מהמשתמש.

רשימת המתודות של המחלקה:

מתודה	משמעות
`HumanPlayer()`	בנאי, מגדיר שחקן חדש
`void playTurn(Board board, Mark mark)`	מבקשת קואורדינטות מהמשתמש וממקמת את הסימן במידה והקואורדינטות "טובות" - כלומר תקינות ולא תפוסות; במקרה והקלט לא תקין, היא תדפיס על כך הודעה ותצפה לקלט חדש (אופן ההדפסות יתואר בהמשך). קואורדינטות חוקיות: טווח השורות והעמודות : $[0, Board.Size - 1]$ דוגמא לקלט חוקי: "31"≡ שורה רביעית, עמודה שניה. (כלומר בוחרים את 3 עבור השורה, ואת 1 עבור העמודה) הערה: ניתן להניח שהקלט מתקבל בצורה תקינה (מספרים טבעיים), אך לא ניתן להניח שהערכים נמצאים בטווח המצוין לעיל.

<u>אופן ההדפסות והנחיית השחקן:</u>

- כדי לבקש קלט, המחלקה תדפיס בהתאם לmark של השחקן: (שימו לב, יש רווח אחד לאחר ה הנקודותיים בכל ההדפסות הבאות)

Player X, type coordinates:
או
Player O, type coordinates:

- אם השחקן בוחר קואורדינטה לא טובה, הדפיסו (שימו לב שבסוף ההדפסה תהיה ירידת שורה):

Invalid mark position, please choose a different position.
Invalid coordinates, type again:
אין צורך להדפיס מחדש Player <mark>, type coordinates לאחר מכן.

- אם השחקן בחר בתרגיל מיקום שכבר תפוס, הדפיסו (שימו לב שבסוף ההדפסה תהיה ירידת שורה):

Mark position is already occupied.
Invalid coordinates, type again:
אין צורך להדפיס מחדש Player <mark>, type coordinates לאחר מכן.

<u>שימו לב!</u>

- לשם קבלת קלט, **<u>חובה</u>** להשתמש במחלקה `KeyboardInput` שניתנה לכם כחלק מקבצי ההגשה (היא מופיעה ב-moodle).

לקבלת קלט, יש לקרוא לפונקציה `KeyboardInput.readInt()` המחזירה ערך `int` שהקליד המשתמש.

אין להגיש את הקובץ `KeyboardInput`, **ואין** לשנות אותו.

- המתודות הפומביות של השחקנים האוטומטיים זהות למתודות הפומביות של ה-`HumanPlayer` ולכן אנו לא מתארים אותן שוב.
- חייב להיות רווח לאחר כל נקודותיים (":") בהדפסות הפלט (כלומר - הנקודותיים לא יהיו צמודות לקלט שהמשתמש יקליד לאחר ההדפסה).
- כל ההודעות מודפסות כמובן ל-stdout (כלומר, השתמשו ב-System.out).

4.4.2. המחלקה `WhateverPlayer` :

בחירה של משבצת אקראית על הלוח (אם אתם לא זוכרים איך בוחרים מספר אקראי, חזרו לשיעור 1.2 בקמפוס).

4.4.3. המחלקה `CleverPlayer` :

לבחירתכם: אסטרטגיה שחכמה יותר מהשחקן `WhateverPlayer` ומנצחת אותו ברוב הפעמים. בדקו את הצלחתו על ידי הרצת טורניר בין $10,000$ משחקים בין השחקן הנוכחי לבין `WhateverPlayer`, על לוח ורצף ניצחון בגדלים הדיפולטיביים. על המימוש שלכם ל `CleverPlayer` לנצח לפחות ב 55% מהמשחקים.

4.4.4. המחלקה GeniusPlayer :

אסטרטגיה שמנצחת את השחקן החכם ברוב הפעמים.

גם כאן, עליכם לוודא את הצלחתו באופן הבא, בטורניר בין $10,000$ משחקים, **על לוח ורצף ניצחון בגדלים הדיפולטיביים:**

- בין השחקן הנוכחי לבין `WhateverPlayer`. על המימוש שלכם ל `GeniusPlayer` לנצח לפחות ב 55% מהמשחקים.
- בין השחקן הנוכחי לבין `CleverPlayer`. על המימוש שלכם ל `CleverPlayer` לנצח לפחות ב 55% מהמשחקים.

4.5. המחלקה `Game` :

מופע של המחלקה מייצג משחק יחיד. עליו לדעת מתי המשחק נגמר, מי היה המנצח והאם הוא הסתיים בתיקו.

מתודה	משמעות
`Game(Player playerX,Player playerO, Renderer renderer)`	בנאי, מגדיר משחק חדש, עם ערכים דיפולטיביים.
`Game(Player playerX,Player playerO, int size, int winStreak,Renderer renderer)`	בנאי נוסף, מגדיר לוח בגודל $size$, ורצף ניצחון באורך $winStreak$. $winStreak$ מהווה את הרצף שצריך (באלכסון, בקו מאונך או מאוזן) כדי להשיג ניצחון. לדוגמה: אם גודל הלוח הוא 4, כלומר 4x4, וה $winStreak$ הוא 3, אזי לוח מנצח יכול להיראות כך: X X X O O (שחקן X ניצח כיוון שיש לו רצף של 3 בשורה הראשונה) במידה שהוכנס ערך לא חוקי (למשל יותר מאשר גודל הלוח, או רצף הקטן מ-2), רצף הניצחון יוגדר להיות שווה לגודל של הלוח.
`int getWinStreak()`	מחזירה את אורך רצף הניצחון
`int getBoardSize()`	מחזירה את גודל הלוח
`Mark run()`	מריצה מהלך של משחק - מתחילתו ועד סופו, ומחזירה את הסמן של המנצח. המשחק מסתיים כאשר לאחד מהשחקנים יש רצף ניצחון או כאשר לא נותרו משבצות ריקות בלוח. במקרה שבו המשחק נגמר בתיקו יוחזר $Mark.BLANK$ **<u>שימו לב!</u>** מיד לאחר כל תור, יש לקרוא למתודה $renderBoard$ של הrenderer להצגת הלוח. גם אם יש מנצח, אנחנו מצפים שקודם יוצג הלוח, ולאחר מכן תכריזו על המנצח

4.6. המחלקה `Tournament` :

תפקיד המחלקה `Tournament` הוא להריץ טורניר של מספר משחקים.

המחלקה מבצעת סדרה של משחקי איקס עיגול (סיבובים) בין שחקנים מסוימים בממשק רינדור מסוים, כאשר:

בסיבוב הראשון, השחקן הראשון משחק איקס והשני עיגול, ובסוף כל סיבוב הם מחליפים סימנים.
באופן הזה, בסבבים עם אינדקס זוגי השחקן הראשון איקס, ואילו באינקדסים האי-זוגיים זה הפוך.

בסיום כל טורניר, מודפסת על המסך התוצאה העדכנית, באופן הבא:

```
######### Results #########
Player 1, [player_type] won: _ rounds
Player 2, [player_type] won: _ rounds
Ties: _
```

לדוגמא:

```
######### Results #########
Player 1, clever won: 687 rounds
Player 2, whatever won: 271 rounds
Ties: 42
```

שימו לב! יש להדפיס הודעת ניצחון רק בסוף הטורניר כולו ולא להדפיס הודעות בסיום כל משחק.

כדי לבצע את תפקידה, המחלקה זקוקה רק לשיטה אחת, מלבד הבנאי. המחלה גם תכיל את המתודה main של התוכנית.

ה API של המחלקה:

משמעות	מתודה
בנאי	**Tournament(int rounds, Renderer renderer,, Player player1, Player player2)**
נקראת ע"י main. במתודה זו תתרחש כל הלוגיקה של הסיבובים וקריאות למשחק. שמות השחקנים הם הסוגים שלהם, והם נמצאים בארגומנטים של שורת ההרצה $args[4]$, $args[5]$	**void playTournament(int size, int winStreak, String playerName1, String playerName2)**
The main method	**Public static void main(String[] args)**

4.7. מפעלים

המפעלים יהיו אחראים על יצירת השחקנים וממשקי הרינדור. בעזרתם אנו אוכפים את "עקרון האחריות הבודדת". זאת דרך פשוטה ואלגנטית להשאיר לעצמינו את האפשרות להוסיף בעתיד עוד שחקנים ואפשרויות רינדור נוספות, כשכל מה שצריך לשנות הוא המפעל המתאים ולא שום חלק אחר בקוד.

4.7.1. `PlayerFactory`

המפעל `PlayerFactory` אחראי למפות את המחרוזת משורת הפקודה לאובייקט שחקן ממשי.

הAPI של המחלקה :

מתודה	משמעות
`PlayerFactory()`	בנאי.
`public Player buildPlayer(String type)`	תיצור ותחזיר טיפוס מסוג השחק המתאים לפי המחרוזת type.

4.7.2. `RendererFactory`

המפעל `RendererFactory` אחראי למפות את המחרוזת משורת הפקודה לממשק הרינדור המתאים.

ה-API של המחלקה:

מתודה	משמעות
`RendererFactory()`	בנאי.
`public Renderer buildRenderer(String type, int size)`	תיצור ותחזיר טיפוס מסוג הרינדור המתאים לפי המחרוזת type.

5.הנחיות והערות:

- הטסטים יבחנו את התרגיל שתגישו בטורניר של כ-10,000 משחקים מעל לוח בגודל 4, ועם רצף ניצחון של 3:
  - מצופה שלאחר 10,000 משחקים `CleverPlayer` ינצח את `WhateverPlayer` ברוב מוחלט של המשחקים (55% מהמשחקים).

- ➢ מצופה שלאחר 10,000 משחקים GeniusPlayer ינצח את CleverPlayer ברוב מוחלט של המשחקים (55% מהמשחקים).

- ❖ זכרו לעבור על מסמך coding-style הנמצא באתר הקורס. ההנחיות במסמך מחייבות.
- ❖ בחרו שמות אינפורמטיביים עבור המשתנים, מתודות, קבועים והודעות ההדפסה.
- ❖ אין לשנות את ה-API של <u>הממשקים</u> שהוגדרו בהוראות התרגיל.
- ❖ ניתן להגדיר פונקציות עזר ושדות פרטיים או קבועים לפי הצורך.
- ❖ אסור להשתמש במגדיר נראות כמו default או protected. כלומר במהלך התרגיל מותר להשתמש רק ב private ו public.
- ❖ כל הקוד הפומבי בתרגיל צריך להיות מתועד היטב (מחלקות, מתודות ושדות).
- ❖ כדאי להתחיל במימוש המחלקות הנצרכות להרצת משחק יחיד, לאחר מכן אנו ממליצים להריץ משחק יחיד בין שני שחקנים אנושיים וכך לבדוק את עצמכם תוך כדי עבודה - מתכנת טוב הוא מי שתופס את הבאגים מוקדם ולא מי שמתכנת בלי באגים.

6.הוראות הגשה

עליכם להגיש קובץ jar/zip/tar בשם $ex1$ (סיומת בהתאם) המכיל את הקבצים הבאים:

.6.1 `Game.java`

.6.2 `Mark.java`

.6.3 `Board.java`

.6.4 `Tournament.java`

.6.5 `Player.java`

.6.6 `Renderer.java`

.6.7 `PlayerFactory.java`

.6.8 `RendererFactory.java`

.6.9 `HumanPlayer.java`

.6.10 `WhateverPlayer.java`

.6.11 `CleverPlayer.java`

6.12. `GeniusPlayer.java`

6.13. `VoidRenderer.java`

6.14. קובץ ה `README`

- בשורה הראשונה בקובץ יופיע שם המשתמש CSE שלכם
- בשורה השניה מספר תעודת הזהות.
- השורה השלישית ריקה.
- בנוסף, ענו בקובץ על השאלות הבאות לפי הסדר:

1. פרטו מהי האסטרטגיה שמימשתם עבור כל אחד מהשחקנים האוטומטיים.
2. הסבירו - מה היתרון בעיצוב התוכנה באופן שבו כל אחת ממחלקות השחקנים מממשת ממשק משותף? ציינו בתשובתכם על איזה עמודי תווך של OOP מתבסס העיצוב.

7.בדיקת ההגשה

7.1. לאחר שהגשתם את התרגיל בתיבת ההגשה הקוד שלכם יעבור טסט presubmit ויווצר הקובץ submission.pdf.

7.2. וודאו שהקובץ נוצר בהצלחה.

7.3. הקובץ מכיל פלט של הטסט המוודא שהקוד שלכם מתקמפל, ומפרט על שגיאות בסיסיות. השתמשו בפלט שבקובץ על מנת לתקן שגיאות בתרגיל שימנעו מאיתנו להריץ את הטסטים הסופיים. (זהו טסט קדם הגשה ולא הטסט הסופי של התרגיל.)

7.4. ניתן להגיש את התרגיל שוב ושוב ללא הגבלה עד למועד ההגשה. (ההגשה האחרונה היא ההגשה הסופית.)

7.5. **שימו לב:** על פי נהלי הקורס חובה לעבור את הטסט ה presubmit ללא שגיאות. קובץ הגשה שלא עובר בהצלחה את הטסט יקבל ציון 0 ולא ייבדק!

7.6. ניתן לחלופין להריץ ישירות את ה presubmit על ידי הרצת הפקודה הבאה (במחשבי בית הספר):

```
~oop2/ex1_presubmit <path to your file>
```

שימו לב שפקודה זו **לא מגישה** את התרגיל בפועל אלא רק מריצה את ה presubmit.
חובה לעבור תמיד גם על הפלט של submission.pdf לאחר ההגשה בתיבת ההגשה לוודא שהכל תקין!

בהצלחה!